A Linked Data Based Messaging Architecture for the Web of Needs

Florian Kleedorfer*,[a], Christina M. Busch[a], Christian Pichler[b], Christian Huemer[b]

[a] Studio Smart Agent Technologies, Research Studios Austria FG, Thurngasse 8/2/16, 1090 Vienna, Austria
[b] Institute for Software Technology and Interactive Systems, Vienna University of Technology, Favoritenstrasse 9-11, 1040 Vienna, Austria

Abstract. *Electronic marketplaces are built to resemble real marketplaces structurally. Consequently, they are centralized systems, walled gardens with an intrinsic tendency to lock merchants and clients in. We argue that this structure is not necessary on the Web and that all online marketplaces could merge into one global medium for exchange. In this paper, we propose an architecture for such a medium based on semantic Web standards, encompassing the functionalities of publishing an intention to buy or sell, finding transaction partners, and conducting transactions. We focus on the basic protocol layer and explain how messaging and linked data are combined in a novel way to realize a highly dynamic communication system.*

Keywords. Electronic Marketplaces • Linked Data • Messaging

Communicated by B. Hofreiter and E. Proper (Guest Editors). Received 2015-06-22. Accepted after 2 revisions on 2016-05-11.

1 Introduction

The World Wide Web was designed as a general information medium. At its inception in 1990, the guiding idea was to build an open network of documents that could be linked to each other. At the beginning, manually curated directory pages were used to discover documents; however, as the Web grew in size, these became impractical to use and even harder to maintain. The situation gave rise to the idea of applying the methods of information retrieval, hitherto mostly applied to well-controlled collections (Brin and Page 1998), to documents on the Web in the form of search engines such as WWWW (the Worldwide Web worm, see McBryan 1994). These services had been the missing ingredient that allowed the Web to scale beyond human manageability while remaining generally usable. Search services made all the difference as they let users structure the Web's content spontaneously according to their needs.

The World Wide Web evolved into more than an information medium: creators of video, music, and of other pieces of art transformed it into a cultural medium; social networking services made it a social medium; e-commerce services made it a medium for business. Its development has been pervasive; people increasingly use the Web to satisfy all kinds of needs. Through all these transformations, it has kept its original bipolar form, a symbiosis of producers and consumers of documents connected by search engines. This form is probably not a coincidence, as it functions like an abstraction of a marketplace, where

* Corresponding author.
E-mail. fkleedorfer@researchstudio.at
This work was supported by the Austrian Federal Ministry of Science and Research and the Austrian Research Promotion Agency (FFG) in the BRIDGE project 'Web of Needs Infrastructure (WIN)' and the COIN project 'Usability, Scalability and Security on the Web of Needs (USS WON)'. Moreover, this work includes preliminary results of scalability tests conducted by Heiko Friedrich and Soheil Human.
Note: This article extends an earlier conference paper, cf. Kleedorfer et al. (2014).

physical objects are on display on different kiosks while clients have to look for things that satisfy their needs, and eventually buy them.

Marketplaces have been known to man for several thousand years. Their historically consistent overall form, a collection of kiosks with goods being offered, and many independent clients buying there, is due to the fundamental asymmetry at the basis of all trade: the asymmetry between its two poles, between supply and demand. Although, conceptually, supply and demand can be said to be of the same kind, namely intentions of taking part in a resource transfer, they differ substantially. Supply appears as the thing being offered, it is thus concrete, can be searched and found, measured and compared. Individual demand, in contrast, denotes the absence of something, thus it does not appear as a physical thing in a traditional marketplace. The same asymmetry pertains to supply and demand on the Web: supply is mostly explicit, has its own representation, such as a product landing page, whereas demand is rarely published as an individual resource. The work at hand aims for reducing this asymmetry.

The first step in this direction is to represent both, supply and demand, explicitly as distinct objects on the Web, and to provide a standardized way for them to communicate. Given these prerequisites, the problem of finding a suitable partner for a resource transfer is equivalent to finding a suitable combination of one supply and one demand object. The search for such combinations can be performed manually or automatically by participants from the demand side as well as from the supply side. Moreover, independent entities can assume the role of a matchmaker and inform the participants involved. In order to allow for independent matchmaking, a standardized way of informing supply and demand objects of a possibly interesting combination is required.

This set of functions (publishing the intention, performing search or matchmaking, and establishing a communication channel) provides a symmetric base for resource transfer on the Web. This set is minimal and necessary, as no function could be removed without losing the overall functionality. This set is also sufficient for a very simple marketplace in which participants only request and make donations, and communicating in natural language is deemed sufficient for the coordination of the transactions. Any other aspect, such as payment, or management of returns, that may be required for more elaborate situations, can be realized based on the established communication channel, either in natural language or by additional technical communication protocols that are beyond the scope of this work. With these additions, defined as open specifications, resource transfer is directly integrated in the Web's infrastructure, which we would then refer to as the Web of needs (WoN).

The contribution of this paper is the detailed explanation of the basic protocol layer, given in Sect. 4. This extension over a previously published work by the same authors (Kleedorfer et al. 2014) is the motivation for changing the title from the name of an argument for the adoption of the Web of Needs to a high-level specification of its communication protocols.

This paper is organised as follows: we first motivate the design of the WoN infrastructure with a simple business case in Sect. 2. Design decisions are discussed in Sect. 3 along with an overview of the architecture they lead to. In Sect. 4, we focus on the basic protocol layer, after which we describe the current state of our prototypical implementation in Sect. 5. Finally, we give an overview of related work in relevant domains in Sect. 6.

2 Motivation

The business case used as a running example is e-commerce and delivery of goods, as illustrated in Fig. 1.

In such a setting, *suppliers* intend to sell and *requestors* intend to buy commodities. A process of discovery and selection leads to a transaction involving requestors, suppliers, and goods or services. The abstract view is illustrated in the upper section of Fig. 1, showing the example business case of a requestor seeking a book and multiple

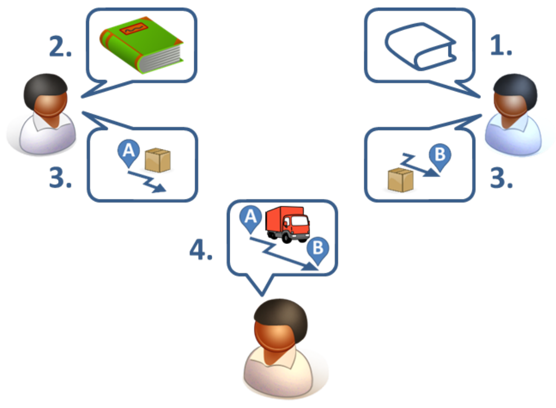

Figure 1: Overview of the running example. The numbers indicate ordering. First, a person expresses the wish for a specific book (1). Another person offers that book (2). They negotiate and agree on a price for the book (not depicted here). Next, the seller expresses the wish for the book to be fetched at location A, the buyer expresses a corresponding wish to receive the book at location B (3). Learning these needs, a person operating a delivery service offers the transport (4), which is eventually carried out (not shown here).

suppliers offering books that may be of interest to the requestor.

In the following, we develop an appropriate representation of this business case for a Web based system, then we discuss consequences this representation entails as a basis for major design decisions. We concentrate on three main steps a user performs: describing their supply or demand, identifying trading partners, and conducting a transaction.

2.1 Modelling Supply and Demand

Description. The view initially described uses roles of people or organisations as elements of the representation: the supplier is used to represent the offer, the requestor is used to represent the demand. For the sole purpose of organising transactions in a marketplace, this description is overly complex: involving the notion of a person evokes many associations that are irrelevant to the purpose, such as gender, age, etc. More importantly, it incurs unnecessary constraints regarding identity: when considering individual transactions, viewing them as occurring between people induces a relationship between the transactions that involve the same people. Such relationships may be interesting and exploitable in many ways but they should only be added when needed. For example, when making recommendations or building a track record of a user's transactions, this information is important. In the general case, though, it is not. In the depicted example, age and gender of the requestor is irrelevant for conducting a transaction, and so are other things they are currently looking to buy. By choosing to disregard the person and focusing on the intention, the irrelevant can be ignored.

For identifying the right abstractions, we consider that prior to entering a transaction, each participant must have developed a mental representation of the transaction and the reason for entering it: in our example, the idea of wanting to sell or buy a book. These abstractions are suitable for representing transactions in a software system: they pertain to exactly one plan or goal and they do not incur unnecessary relationships to other notions or transactions. In our example, the requestor has to develop the idea that they intend to buy a book. This idea may identify a specific book or encompass certain properties it should have (e. g. a cookbook for Italian food). Likewise, the suppliers develop the idea to sell a specific book.

Reified as objects in the system, such representations are used to identify the transaction's endpoints, serving as proxies for their respective creators or *owners*; we will refer to them as *owner proxies* (or proxies for short). These objects contain a description of the reason for entering a transaction (i. e. supply or demand) and allow their owners to connect to other objects of this kind and communicate. This view is depicted by the lower portion of Fig. 1, where we see that the requestor controls a proxy representing their demand for a book and the suppliers control proxies representing their offers.

When describing a commodity, one chooses a level of granularity compatible with the context in which the description is needed. When the commodity, the requestor, and the supplier are physically present, and the type and conditions of the transaction are prescribed by the circumstances, descriptions are mostly unnecessary—as is the case in traditional public marketplaces. In electronic marketplaces, the same commodities may be described with the highest level of detail. A generally useful marketplace solution must be flexible enough to support both cases equally well, therefore users should be given the options to describe their intentions in unstructured form using properties such as title, description, and tags. Moreover, there should be support for the input of generally useful properties such as time of availability, price information, and location information. For expressing fine-grained properties, it should be possible to use an appropriate data structure. The example use case may occur in an informal setting, as may be encountered when trading used books. In such a setting, using title and description may be enough. The owner proxies representing an industrial publisher's books may contain very detailed domain-specific data such as ISBN, year, and other properties.

Identification of combinations. The representation of intentions developed above allows for matching supply and demand based on their descriptions. Such matching can either be done manually, similar to web search, or it can be done automatically, applying appropriate algorithms and possibly a large body of formalized background knowledge. The latter approach raises the question how exactly such a matching algorithm should work, given the fact that customers may describe their demand differently than suppliers describe their offers. While the specific approaches for automatic matching are beyond the scope of this work, the proposed infrastructure is designed to allow for them to be improved gradually, by providing information about historic supply/demand combinations and about past suggestions of such combinations in the form of open data.

Communication between proxies. When a suitable combination of owner proxies has been identified, it is in the interest of both parties to establish a communication channel in order to negotiate. It may be sufficient to use informal natural language for such negotiations. In other settings, the preferred way might be to apply formalized protocols (for instance, according to the WS-BA specifications, see Robinson and Newcomer 2009). Therefore, defining one protocol for all such situations is unrealistic. The minimal functionality, has two aspects: first, a handshake between the participants that, if successful, leads to the creation of a communication channel. Second, the participants need to be able to exchange messages, which must be extensible to support specialized protocols, but provide for the simple case of informal conversation equally well.

2.2 Consequences

Symmetric Representation. The most important property of the model described here is that both participants of a transaction represent their intention as a distinct object. Thus, they have the option to be the active (initiating) as well as the passive (accepting) part during the establishment of a connection—hence the situation becomes symmetrical. Moreover, this allows for a separation of concerns: that of specifying the intention and that of finding transaction partners. By choosing a service for specifying their supply or demand, users are not limited to that same service for identification of transaction partners—to the contrary, any matching service can suggest such partners. In the illustrating example, the requestor is not required to initiate the transaction and she is not limited to the suggestions of the service she used for publishing her demand. Instead, the wish for a book is represented by an owner proxy that can be found by any matching service and can be contacted by other users.

Compositions. A more subtle consequence of the objectification of both supply and demand in the form of owner proxies is that compositions become possible on both sides. In the general case, the supply side defines and offers a commodity in the hope of meeting demand. The commodity may be a composition of goods and services, but it is intrinsically an atomic entity: if for some reason only a part of that bundle of goods and services were to be sold, that part would become a commodity in its own right; the same is true for the composition of commodities. However, there are costs involved in the creation of new commodities through composition or decomposition. This effort is only made if there is sufficient demand. All-inclusive holiday offers are a well-known example of such complex commodities. As objectification becomes possible on the demand side, it is no longer futile to express demand for combinations of commodities that do not exist yet because each element can be obtained from a different supplier. Moreover, such complex demand specifications are valuable in themselves as they encapsulate knowledge about how a certain goal can be reached through a combination of goods and services, and they can be re-used and re-mixed. Our book-selling example may not justify such complexity, but consider planning a wedding. It requires finding and coordinating catering, musicians, a suitable location, preparation of invitations, and probably much more, all of which must be available at certain, inter-dependent dates and in inter-dependent quantities. Such a plan could be represented as a composition of rather simple owner proxies that are tied together with relationships expressing these constraints. The first formulation of such a plan might require expert knowledge and take a lot of time, but subsequent instantiations may not require quite as much expertise. The frequent application of the same plan may lead to incremental improvement and the development of different versions for different scenarios.

3 Architecture

Having derived the high-level view from the main business case, we describe the approach for its realization. We begin by explaining design decisions, then we give an overview of the architecture.

3.1 Design Decisions

In an earlier work, the following non-functional requirements for the Web of needs were defined: access, usability, fairness, simplicity, scalability, timeliness, robustness, security, privacy, and/or anonymity (Kleedorfer and Busch 2013). Together with the results from the previous section, these are taken into account in the design we develop in the following.

De-centralization. As stated before, privacy is essential, as is the neutrality of the infrastructure so as to ensure fairness. Both are much less of a problem in de-centralized systems than in centralized ones, as there is no single party with access to all the data and no opportunity for one party to control access. Moreover, the cost of running and growing a de-centralized infrastructure is naturally shared among those operating nodes of the network, so no or only little initial centralization of capital is required to establish such an infrastructure. Consequently, we envision the Web of needs as a network of nodes, which we call *WoN nodes*.

RDF-based description language. The description of intentions represented by owner proxies can take a variety of forms. The formalism for expressing these descriptions in machine-readable form must be highly flexible, allowing for arbitrary structured data with clear semantics. RDF (Manola and Miller 2004) allows for expressing arbitrary data structures and provides options for sharing schema (ontology) and data in an unambiguous and standardized way. For these reasons, RDF is chosen as the basic data model for the WoN infrastructure. In order to allow for identification of communication partners by independent services, the descriptions are made publicly available as linked data.

Public connection information. Whenever users decide to connect with each other, their owner proxies follow a simple protocol that leads to a *connection* being established. When a connection is established between two owner proxies, each one creates an object representing that connection and publishes it as linked data, which means that it is publicly shown which proxies are connected and what the state of their conversation is.

Always on-line nodes. Retrieval of information about owner proxies must be possible at all times to allow for asynchronous communication and matchmaking. Therefore, desktop computers or mobile apps cannot host their owner proxies themselves. Rather, for each owner proxy, they may choose from a set of available WoN nodes and use a standardized protocol to create an owner proxy there. The WoN nodes are servers dedicated to the task of hosting owner proxies and are always on-line. Thus, users can create different owner proxies on different nodes, achieving a minimum of privacy by distribution.

Thin clients. In order to allow for controlling a user proxy intermittently from different applications (e. g. a mobile application and a Web site), all information is kept on the WoN node hosting the proxy. As all relevant information is available as linked data, *owner applications* (i. e. clients) do not need to keep track of all events pertaining to their proxies; whenever needed, local information can be refreshed, provided that the application can authenticate itself as the proxy's legitimate owner.

Message-oriented communication. Owner applications cannot be assumed to be on-line all the time. However, activity pertaining to a user's proxies while the user's owner application is off-line may occur on both, the server side as on the client side. On the client side, a user may want to write a message to a communication partner; on the server side, a new match might just have been received from a matching service. Because of this inherent asynchronicity, the message-oriented approach is far better suited for this infrastructure than remote procedure calls. In order to allow for flexibility in terms of message composition and so as to have a unified data layer holding the complete application data, the messages are expressed in RDF and made available as linked data.

Identity. In order to serve as an endpoint for reliable transactions, an owner proxy must be able to prove its identity to others. This is achieved by integrating cryptographic technology by using WEBID (Sambra et al. 2013).

Mutable and immutable data. In our distributed setting, it is important to make clear which information about an owner proxy can be expected to change over time and which part of its description will remain unchanged. For example, if an owner proxy represents a taxi cab, information about its pricing should be marked as *static*, whereas information about its current location and availability should be *transient*. Any interested party needs to be able to verify if the static information is indeed unchanged so as to avoid being tricked. On the other hand, of course, information marked as static does change over time, in which case it is required that interested parties are informed of this fact and can choose whether to continue their relationship with the owner proxy in question or not.

3.2 Architecture Overview

The above design decisions lead to the architecture we describe in the following, starting with the big picture. In the bird's eye view, the Web of needs consists of three types of elements, as depicted in Fig. 3. The basic functionality—allowing CRUD operations on owner proxies, establishing connections between them, and receiving hints from matching services—is provided by WoN nodes. *Owner applications* allow users to create and control their proxies. *Matching services* crawl the proxies' data found on WoN nodes and send *hint* messages to owner proxies.

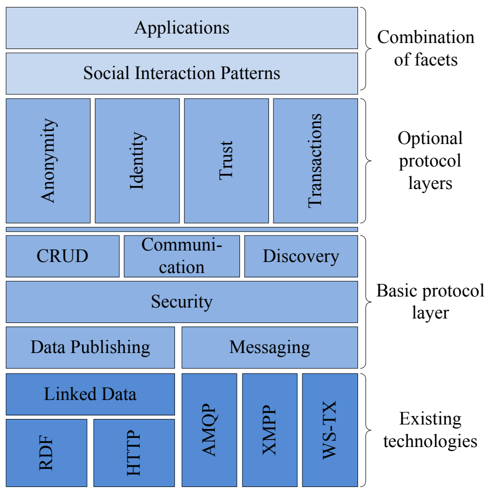

Figure 2: Diagram illustrating the layers in the protocol and technology stack. The basic protocol layer, second from the bottom in the illustration, is described in detail in Sect. 4

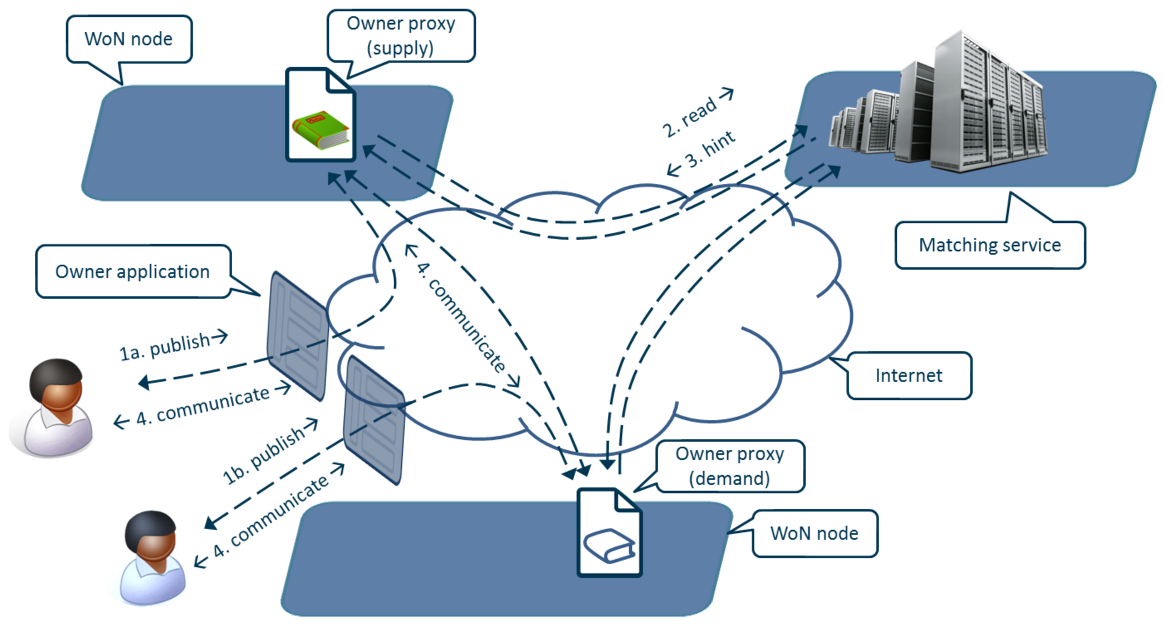

Figure 3: Communication and deployment diagram illustrating how users publish their intentions as owner proxies (1a, 1b) on different WoN nodes, using their owner applications, and how a third-party matching service reads the descriptions (2) and helps establishing a communication channel by sending both owner proxies a hint message (3) through which users communicate (4). In our example use case, they agree to the transfer.

The goal of our work is to define an architecture for an open, de-centralized worldwide marketplace. In order to reach this goal, existing technologies are arranged so as to provide a basis for end-user oriented applications. Fig. 2 illustrates the most prominent base technologies in the bottom layer. The basic protocol layer provides secure CRUD operations, communication between owner proxies and discovery of suitable communication partners. The optional layers provide strong anonymity, standardized links to identities, inclusion of trust mechanisms, and different flavors of transactionality. On top of this stack, arbitrary facets can be implemented to map important social interaction patterns, which in turn can be combined in end-user facing applications.

4 Basic Protocol Layer

The overall architecture is based on the messaging paradigm (Hohpe and Woolf 2004). Messages are expressed according to the RDF data model. Each message is sent to a WoN node where it is stored and handed to a processor depending on addressing information. The message processor is responsible for state changes of the addressed entity and may forward the message to another WoN node or one or more owner applications. The fundamental structures to represent owner proxies and their interconnections are depicted in Fig. 4. The main classes are `OwnerProxy` and `Connection`. The latter represent the M:N relationships between owner proxies in such a way that for each relationship, each of the two owner proxies maintains a dedicated connection object to represent its side of the relationship. It is important to note that this view only shows the 'local' view for one owner proxy; the fact that it may be connected to another owner proxy is solely represented by its relationship with the `Connection` class. This design allows for two owner proxies to reside in different systems without limiting their ability to interact.

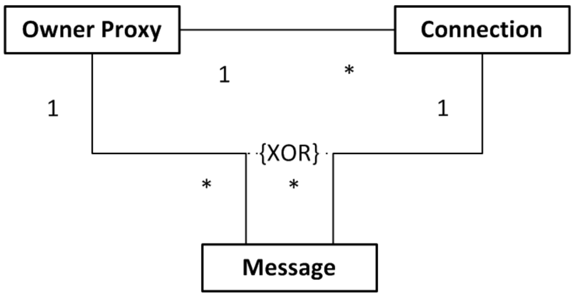

Figure 4: Class diagram illustrating the basic entities and their relationships

Each owner proxy as well as each connection object has its own message processing facility. When sending a message, the URI of the destination (either owner proxy or connection) is specified as the recipient of the message. In other words, it is possible to direct a message at an owner proxy or at a connection. Addressable objects have a *message container* (not shown in the diagram) for messages that were sent or received by the object, all of which are made available as linked data. In addition to that, the WoN node has a general message processing facility that can be addressed, for example, to create new owner proxies (see below).

In order to ensure the authenticity of messages, each owner proxy contains or references a public key used for asymmetric cryptography, and all messages are signed with the corresponding private key. This approach is chosen so as to enable third parties to verify the authenticity of all data visible to them in the system and to require as little trust as possible in any of the entities involved in a message exchange. Consequently, clients must be able to manage their private keys. Moreover, they must be capable of producing valid RDF datasets and of creating and verifying signatures.

4.1 WoN node descriptor

Clients obtain information about the WoN node by retrieving its *descriptor*, which is available as linked data. The descriptor contains the WoN node's public key and URI prefixes that are to be used when generating new message URI or new owner proxy URI. Moveover, it specifies how WoN messages can be sent to and received from the node. Any duplex transport is conceivable for this purpose (such as JMS or Websockets), and the supported protocol description contains the transport-specific connection details such as broker URI, message queue names etc. In addition to that, the descriptor lists the facets supported by the WoN node.

4.2 General message processing

The messages exchanged in the system are represented as RDF datasets (Zimmermann 2014). In order to enable unified data retrieval, the messages are published as linked data. The RDF dataset is used as it allows for the specification collections of RDF triples called named graphs. These collections can be given unique identifiers, which makes it possible to state facts about them and add these facts to another named graph in the dataset. This construction is used, for example, to separate addressing information from the message content. Moreover, it allows for collecting triples in one graph, creating a cryptographic signature of the content, and representing the signature in another graph.

When a client intends to send a message to an entity residing on a known WoN node, it performs the following steps:

1. Generate a new RDF dataset.
2. Generate an URI as the 'main' resource to be used by the WoN node, and set this URI as the base URI of the dataset. The URI pattern to be used for the main resource is specified in the WoN node descriptor.
3. Generate and populate one or more RDF graphs representing the content of the message (*content graphs*).
4. Generate a cryptographic signature for each content graph.
5. Add one *signature graph* for each signature to the dataset, describing the signature according to the specification of the Signing Framework (Kasten et al. 2014). This includes a reference to the public key used for signing.

6. Generate an *envelope graph* containing
   a) the type of the message.
   b) a reference to the recipient URI.
   c) a reference to the description of the WoN node that hosts the recipient. This could be omitted as it can be looked up by dereferencing the recipient URI. However, the WoN node URI is required at numerous steps during the routing process, so adding it to the envelope simplifies routing.
   d) references to all content graphs along with the signature values.
7. Generate a signature of the envelope graph.
8. Add a signature graph describing that signature to the dataset.
9. Serialize the dataset and send it to the WoN node via one of the transports specified in the WoN node's descriptor (see Sect. 4.1). Current options for serialization are all formats that allow named graphs, which are JSON-LD (Sporny et al. 2014), TRIG (Bizer and Cyganiak 2014), or N-QUADS (Carothers 2014).

One may argue that Step 2 is problematic as the client may happen to choose an URI that is already used by the WoN node. In this case, the message is rejected, leaving the client only the option to generate another URI, update the message dataset, re-calculate the signatures, and send it again. This behaviour is clearly more complicated than letting the server generate the message URI upon receiving the message. The reason for choosing this more complicated approach is that it leaves no way of replaying the same message or using it in a different context as the signatures tie the content to the URI by which it is accessible via HTTP.

Upon receiving a message, the WoN node executes the following steps:

1. De-serialize the dataset.
2. Check the message URI:
   a) Identify the message URI created by the client.
   b) Check if it is still available and atomically mark it as taken.
   c) Return an error if the URI is already taken.
3. Verify the signatures:
   a) Identify the signature graphs.
   b) Load the public key referred to in the signature graph.
   c) Verify the signature of each signed graph using the public key referenced in it.
   d) Abort with an error if any of these steps fail.
4. Identify the envelope graph.
5. Identify the recipient of the message
6. Dispatch the message to the message processor responsible for the recipient's messages.
7. If the message resulted in the creation of a new object (e. g. a new owner proxy), the following additional steps are taken:
   a) If the new object is an owner proxy the RDF graphs describing it are extracted from the message along with their signature graphs.
   b) A message container and other resources (depending on the type of the object) are generated and this 'technical' information is stored in a separate *system information graph* in the dataset of the new object, along with a graph containing its signature.
8. Add the complete message that was received to the message container of the recipient; if the message resulted in the creation of a new object the message is stored in its message container.
9. Generate a response message:
   a) Perform the required steps to create a message with the sender of the original message as the recipient and vice versa.
   b) Add a reference to the URI of the message that was just processed.
   c) Add the signature value of that message's envelope.
   d) Set the type to ERROR if an error occurred, SUCCESS otherwise.

e) Store the new message in the message container of the recipient.

f) Send the message to the original sender.

4.3 Creating and managing owner proxies

The general message processor of the WoN node understands four message types: CREATE, UPDATE, REPLACE, and DELETE. These messages are used to create, modify, or delete owner proxies. The CREATE message is used to create a new owner proxy; it is described in more detail below. The contents of one or more named graphs inside the RDF dataset describing the owner proxy can be overwritten using an UPDATE message. Such a modification is only allowed if the graphs in question were originally intended to be updated and are marked as *transient* at creation time. If the contents of *static* graphs in the owner proxy's dataset are to be changed, the whole modified dataset is sent in a REPLACE message. A REPLACE message is processed like a CREATE message with the addition that the replaced owner proxy is deactivated and a reference to the new version is added to its description. Likewise, a reference to the old version is added to the description of the new owner proxy. The DELETE message causes the deletion of all data of all versions of the owner proxy from the WoN node. The URI of these objects are marked as deleted by the WoN node, and any subsequent request to these URI is responded to with HTTP status code 410 (GONE); messages addressed at one of these entities will cause an error to be returned. The message exchange for proxy creation is illustrated in Fig. 5.

When a client intends to create an owner proxy on a known WoN node, it performs the steps listed in Sect. 4.2 required for sending a message. At Step 3 the following steps are taken:

1. Create one or more content graphs in the RDF dataset and add triples describing the owner proxy.
2. Create a graph in the RDF dataset and add triples that mark the content graphs as transient or static, thereby defining which ones may later be overwritten by sending an UPDATE message.

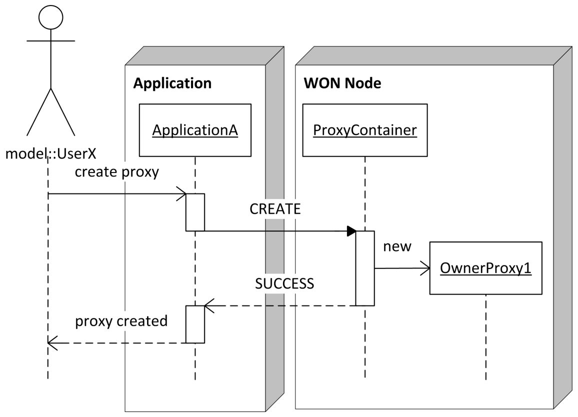

Figure 5: Sequence diagram illustrating the communication when an owner proxy is created by a user.

In Step 6, the message type is set to CREATE and the URI of the new owner proxy is set as recipient. For replacing an owner proxy, the process is the same as for CREATE except for the message type (REPLACE). Updating is done in the same way, the only difference being that only graphs marked as transient are added to the message content, and UPDATE is used as the message type. For deleting an owner proxy and all its versions, a message of type DELETE is sent with the owner proxy's URI as the recipient.

4.4 Establishing connections and communicating

For the owners of two proxy objects to communicate, it is required that one of them sends a CONNECT message on behalf of their proxy to the remote one, which may answer with either an OPEN message, thereby establishing the connection, or a CLOSE message, which denies the connection (or closes it if it is established). Sending the CONNECT message causes a new connection object to be created by the WoN node and added to the owner proxy's connection container; receiving a CONNECT message has the same effect on the receiving WoN node. Both messages include to the owner proxy's URI and the signatures of its static graphs, so that it is made impossible to change data marked as static after establishing a connection. When a message sent in response to another (e. g. an OPEN message sent in response to a CONNECT), the sender who is replying must add a reference to the message they received and include their signature value in the envelope. This ensures that the authenticity of the conversation can be verified. As soon as the connection is established, the communication partners can exchange messages, using the type CONNECTION_MESSAGE. The envelope graph of such messages contains the signatures of the last message(s) in the conversation that have not yet been 'cited' this way. The payload of such messages is RDF-encoded data contained in signed content graphs that are added to the message dataset as in the case of CREATE messages (see above). The message exchange taking place for establishing a connection is illustrated in Fig. 6.

4.5 Discovery

The functionality described so far allows for distributed operation of WoN nodes, creation of owner proxies, and exchange of messages between owners. At this layer of the architecture, no assertions are made with respect to the semantics of owner proxies and connections. While it is conceivable that users create owner proxies just for the sake of communicating with known counterparts, the intended use is that owner proxies contain semantically rich descriptions of their owner's intentions and that independent parties use these descriptions to suggest communication partners. In such a case, a message of type HINT is composed and sent to one or both owner proxies involved. A HINT message has a signed content graph containing the URI of the counterpart, a score indicating the quality of the match, and optionally more triples 'explaining' the match. Upon receiving a HINT, the WoN node creates a new connection object containing references to both owner proxies involved and adds it to the receiving owner proxy's connection container. This step is omitted if a connection object for those proxies already exists. The message itself is added to the connection's message container. The message exchange taking place for discovery is illustrated in Fig. 7.

4.6 Facets

When creating an owner proxy, the user decides which behaviours they want it to support. This is done by adding *facets* to the proxy's description. A facet is a communication protocol on top of the protocol described earlier, used for making connections and exchanging messages. The facet defines the semantics of this message exchange and is tied to dedicated processing logic on the WoN node. Consequently, only facets that are supported by the WoN node can be chosen by users. The guiding idea here is to define a rather simple interface (the facet), the contract of which can be formulated as linked data. Implementations of that contract should be easy to create and add as modules for existing WoN node implementations and owner applications. In the following, we describe some possible facets:

Owner Facet. This is the most basic facet, offering just the standard functionality and adding no semantics. All types of messages received from the connected owner proxy are forwarded to the owner application and there is no requirement as to the content of messages of type CONNECTION_MESSAGE.

Group Facet. An owner proxy with this facet acts as the message dispatcher of a group. A message received on a connection of that facet is forwarded to all other connected owner proxies except the original sender. This allows for implementing a group chat, for example.

Link Facet. The link facet allows for establishing a link between the two owner proxies directly in their RDF content. The owner proxy sending the CONNECT message adds two content graphs to the message: one contains the RDF data that should be added to the remote proxy's content graphs, the other is the one that it will add to its own content graphs as soon as the remote proxy accepts the connection. Upon the closing of the connection, the respective content graphs are removed from the owner proxies' linked data representations. This behaviour enables the creation of arbitrarily complex structures of owner proxies that are expressed with typed links and,

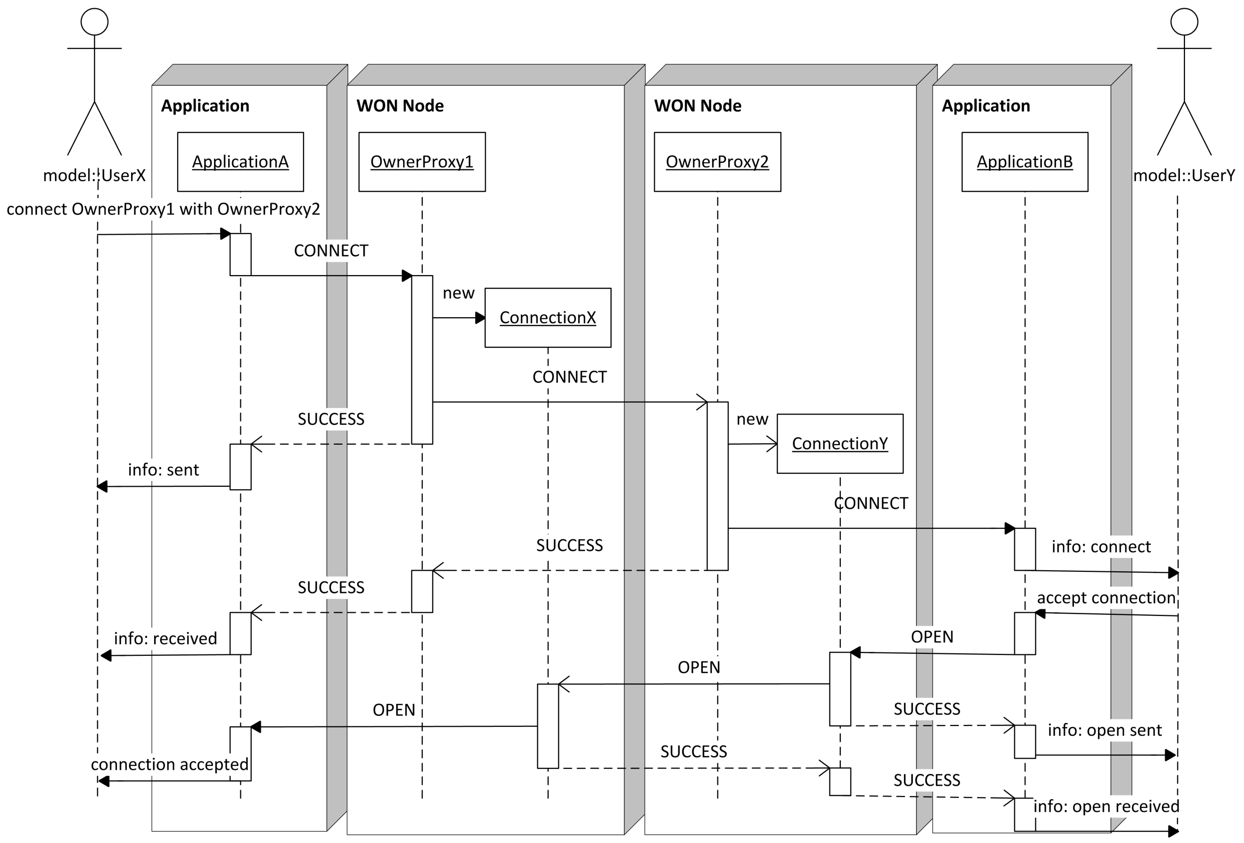

Figure 6: Sequence diagram illustrating the communication taking place when the owner initiates a connection with another owner and the counterpart accepts the connection.

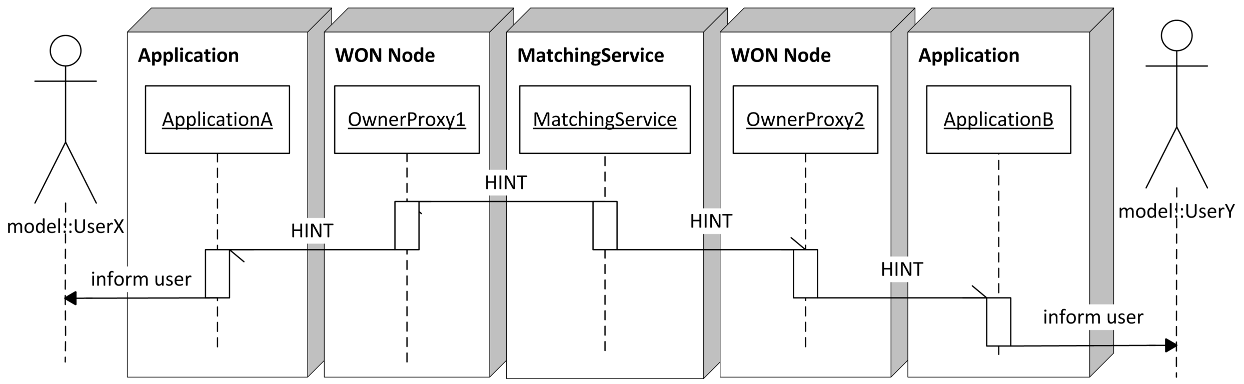

Figure 7: Sequence diagram illustrating the communication taking place when a matching service sends a HINT message to two owner proxies. The connection objects created on the WoN nodes in response to the HINT messages are not shown here.

optionally, additional metadata. As explained in Sect. 2.2, such relationships are necessary for defining compositions of supply or demand. Fig. 8 shows how such links can tie owner proxies together to achieve the functionality required for the running example.

4.7 Novelty and Benefits of the Approach

The reason for not choosing the mature stack of XML based service oriented architecture technologies (Papazoglou 2008) as the basis of the WoN infrastructure is the need for verifiability of the state of communication by third parties and the goal of universal data integration through linked data. During the development of the protocols, it was realized that third parties would sometimes need to be able to verify who said what in a given conversation, for example, to resolve a dispute or to allow verification of the state of a distributed transaction by all participants. This means that for such cases, messages cannot be realized merely as volatile containers for actual payload but have to be stored and made available upon request by authorized parties. RDF and linked data had been chosen as the data model for representing content, and messages having been recognized as part of the user-generated content (and not part of a communication subsystem), were chosen to be represented in RDF as well. In order to achieve that, existing approaches for creating cryptographic signatures of RDF and embedding them in linked data were integrated, leading to an infrastructure that exposes its complete data model as cryptographically secured linked data. As there is no other way to publish data in this infrastructure than to send a message, this essentially constitutes a cryptographically assured audit trail for any piece of data in WoN without requiring centralized identity management. Moreover, linked data provides unified data access, so there is no need for dedicated API for querying different parts of the data model. Moreover, the complete state of a WoN node, including the state of all user transactions, can be retrieved at any time by crawling the WoN node's data, which has potential uses for backups and auditing.

5 Prototypical Implementation

The presented architecture was iteratively developed using an open source design prototype to ensure its technical feasibility. The prototype consists of the following components:

WoN node. This service's main functionalities are i) providing the functionality to create owner proxies in reaction to messages sent by owner applications, ii) creating connection objects between owner proxies in reaction to HINT or CONNECT messages received from owner applications or matching services, and iii) serving the linked data pertaining to owner proxies, connections and messages via HTTPS. The service is implemented in JAVA based on the SPRINGFRAMEWORK. Messaging is realized through the JMS 1.1 (Deakin 2003) implementation ACTIVEMQ; the core data structures (owner proxies, connections, messages) are stored in a relational database. Freely definable RDF graphs that are received as parts of messages are stored as BLOBs in the database. Linked data is constructed on the fly in response to requests.

Owner application. The owner-facing user interface is realized as an ANGULARJS Web application that communicates with its server side through a REST API and a Websocket. The server side of the owner application connects to the WoN nodes required for communication through the WoN nodes' ACTIVEMQ endpoints. The owner application provides the functionality to create and manipulate owner proxies, to view hints and establish chat connections with other users via their proxies. Moreover, users can create accounts on the owner application under which all their proxies will be accessible to them. The cryptographic keys for the proxies are generated and stored in the owner application.

Matching service. Two approaches are implemented for learning about owner proxies: a 'pull' approach through crawling and a 'push' approach through publish/subscribe. Whenever a matching service encounters a new WoN node, it crawls the linked data available from it, and it subscribes to a messaging topic to be informed when a new owner proxy is created. When a new proxy is encountered through one of these methods, its data is stored in an RDF database and an internal event is generated that is consumed by implementations of matching algorithms, which in turn locate the best-matching owner proxies and

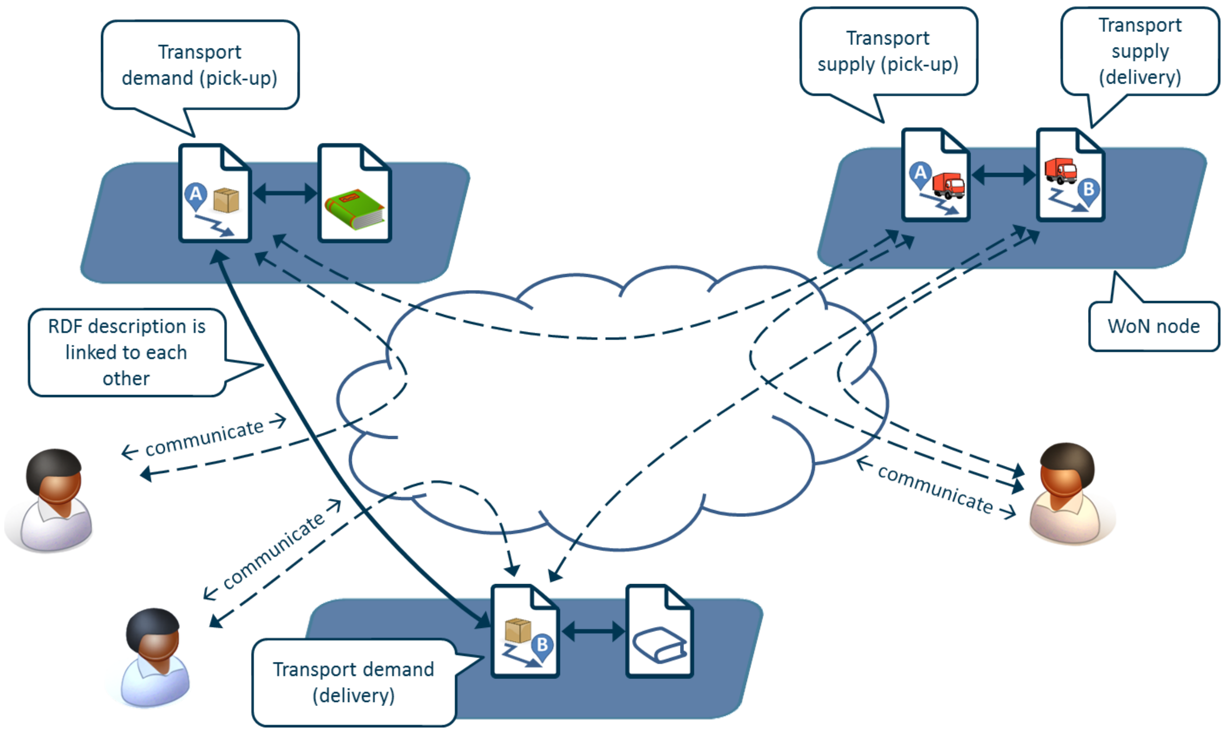
Transport demand (pick-up)
Transport supply (pick-up)
Transport supply (delivery)
WoN node
RDF description is linked to each other
communicate
communicate
communicate
Transport demand (delivery)
A
B
A
B

Figure 8: Diagram illustrating how pick-up and delivery of the book is organised via WoN. Buyer and seller have created proxies for pick-up and delivery, respectively. They have established a connection between these proxies using the link facet (see Sect. 4.6), and hence their descriptions point to each other, allowing third parties to discover both ends of the relationship while both buyer and seller stay in control of their respective end. A delivery service has created proxies offering pick-up and delivery, also linking to each other. A matching service has sent hints to the parties involved, who all accepted to be connected. Now the delivery service has a communication channel to seller and buyer and is able to negotiate pick-up and delivery dates individually. Note that in this depiction, owner applications have been left out for simplicity.

pass this information to a multiplexing component that generates HINT messages and sends them to the WoN node hosting the proxy. This service is realized based on the AKKA actor framework and uses a SPARQL-1.1 (Aranda et al. 2013) compliant RDF database for storing crawled data. Matching is implemented based on the SIREN information retrieval engine based on the SOLR search server.

Bot framework. For conducting tests and as a means for connecting existing applications to the Web of Needs, a framework for programming autonomous WoN agents, called *bots* was developed. A bot can create and controls owner proxies on WoN nodes. It has an event loop that is regularly executed and in addition to that, it receives events whenever a message is received for one of the owner proxies controlled by it. A bot provides a unified integration point for legacy applications where incoming events can be transformed to internal service calls. Bots have been used to simulate user behaviour in integration tests and load tests.

These components are available as DOCKER containers to allow for efficient deployment. For conducting tests, three 3.6 GHz Core2 Duo machines with 4 GB RAM running DOCKER 1.7.0 on UBUNTU LINUX 14.04 LTS were used. Two machines ran two WoN nodes, one owner application and a POSTGRESQL database each, the third ran all required components of the matching system. Functional tests involving proxy creation on different WoN nodes, matching, establishment of connections, and a simulated conversation were successful. Initial stability tests involving the generation of proxies with descriptions randomly chosen from a private email corpus and generating hints for each proxy showed that the system exhibits its initial response times of 0.15 seconds, growing to about 0.5 seconds when proxies are generated at a rate of one proxy every two seconds over a duration of 12 hours, which we deem sufficient as a baseline for conducting user tests. A more detailed performance evaluation is being prepared.

6 Related Work

Having motivated and explained our approach, we now give an overview of related and relevant systems or concepts and compare them to the Web of needs.

E-Marketplaces. To the best of our knowledge, there is no prior work on global, open, web based market infrastructures to learn from or build upon. The well-established and closely related concept of e-marketplaces, though, has been scientifically examined. The approach pursued in the Web of needs, we argue, differs from traditional e-marketplaces insofar as it does not create separate vertical marketplaces for different niches; rather, it creates one unified marketplace on the Web. This helps to lower transaction cost below the current levels in the long run and provides a marketplace for niches that aren't profitable enough for a dedicated Web site to emerge. We do not have estimates for the size of these 'long-tail markets' where supply and demand must be assumed to exist but lack mediation; if the long tail phenomenon translates from other domains to this one, however, it should be considerable.

Traditionally, e-marketplaces have been used in business-to-business (B2B) or business-to-consumer (B2C) context. In addition to B2C and B2B marketplaces, examples of successful consumer-to-consumer (C2C) e-marketplaces have emerged in recent years. These include classified ads portals and auctioning websites like CRAIGSLIST or EBAY as well as specialized platforms such as AIRBNB or COUCHSURFING. In C2C e-marketplaces, individuals are responsible for both offerings and purchases. All of these types of marketplaces are realizable on the basis of the Web of needs infrastructure, which allows transactions that span domains that are currently organised in vertical marketplaces (such as holiday apartment rentals, taxi services, or restaurant bookings), thereby clearly offering additional useful functionality that cannot be provided by any such platform alone, namely the unified access to all the verticals, and hence the possibility to combine transactions, for example, to buy opera tickets, reserve a table in a restaurant, and order a taxi without the burden of switching marketplace.

Very closely related to the Web of needs is the effort of publishing semantically rich offer and demand descriptions on existing e-marketplaces using vocabularies like GOODRELATIONS (Hepp 2008) or SCHEMA.ORG. The main difference is that these vocabularies are tools for describing entities, but they do not define service interfaces; entities suitable for interaction can be found automatically, but there is no standardized way to establish a connection with them.

Intention Economy. The term *intention economy* denotes an environment in which customers use software systems to manage their relationships with vendors (Searls 2012), so-called *vendor relationship management* (VRM) tools. Among other functionalities, such a tool supports expressing demand in the form of a *personal request for proposal* (pRFP); the act of publishing such a pRFP is referred to as intentcasting. The notion of the pRFP is quite similar to our concept of a owner proxy representing a demand, and the technical artifact most similar to the owner proxy is the pico (persistent computing object) in the KYNETIX RULES ENGINE (Windley 2011). A difference is the VRM community's focus on commercial activities, in contrast to which the Web of needs is intended for more general use. It remains to be seen how the Web of needs infrastructure fits in with the tools that are being developed in the VRM community.

Discussion Groups. Countless marketplaces are organised on the Internet in discussion groups.

The technical bases range from Web forum software over mailing lists to groups in social networking Web sites. The main advantages of these groups are simplicity, openness, and the like-mindedness of participants. They are normally self-regulated with respect to a stated or implicit code of conduct and allow postings consisting of text and images (as opposed to structured data). They work best below a certain frequency of messages, because users have to read all messages so as to decide whether they want to trade. If postings are added with too high a frequency, it becomes hard to follow the updates, which reduces the usefulness of the group for individual users, although in principle the higher number of available options for trading should increase usefulness. At some point, such groups tend to split up in multiple smaller ones. Consequently, such approaches lead to a large number of groups, causing users the problem of identification of the right group for a given case. Moreover, there often are a number of redundant groups for the same type of commodity, location etc., in different channels such as FACEBOOK, mailing lists, or dedicated Web sites, making the decision for one of them even more difficult. The Web of needs infrastructure alleviates all of these problems and may prove more useful to users in the long run.

Social Networks. The basic functionalities social networking platforms offer are creating and maintaining relationships between user identities and using them for communication purposes. With respect to user-to-account cardinality as described by Dalton (2013), such platforms mostly try to achieve a one-to-one cardinality, i. e. one user has one account. Of the big players, only TWITTER allows one-to-many and many-to-one cardinalities. Users with profiles on more than one social networking site implicitly have a one-to-many relationship with these profiles, where the platforms set the context for these accounts, resulting in differences in the profile characteristics. For example, LINKEDIN focuses on business relationships whereas FACEBOOK is more about family and friendship; users shape their profiles accordingly. In contrast, based on the facet generalization introduced in Sect. 3.1, social networking functionalities can be built on the Web of needs allowing any user-to-profile cardinality. In such applications, an owner proxy can represent a user; if credentials are shared, or access is delegated otherwise, it can represent many users, allowing one-to-one and many-to-one cardinalities. Users can create separate social networks as needed, for different contexts such as family, friends, political contexts etc. In each such network, the same person has a different identity. These identities can be aggregated by the user as desired.

In the context of social networking, a highly relevant related work is the project on distributed semantic social networking (DSSN) that takes a more lightweight approach for data access and communication than was chosen for the Web of needs (Tramp et al. 2014). While for our architecture, social networking is one of many possible applications, DSSN is specifically designed for it but can be combined with other semantic Web based systems to offer richer functionality.

7 Conclusion

In this paper, we have focused on motivating why the Web of needs as a generic cooperation framework can serve as a basis of a worldwide on-line marketplace. We explain design decisions, and give a detailed explanation of the basic layer of our architecture.

Our main stance is that a worldwide marketplace must be as de-centralized and open as the worldwide Web. In general, transactions should be recorded publicly. The central element of such a marketplace is the owner proxy, an entity anonymously controlled by a user. It contains a description of the task it has been created for, in our running example is the purchase or sale of a book, and encapsulates the least amount of data and functionality that is required to perform the desired task. Owner proxies are made aware of each other by independent matching services that compare their descriptions and inform them about possible transaction counterparts.

An important goal of the work at hand is to provide a communication architecture for any content domain and any application scenario in such a way that these domains and scenarios can blend into each other. It should not be necessary to deploy and configure a server (i. e. WoN node) specifically for one content domain or application scenario. The architecture presented here exhibits these features. Users are free to specify any information for the content of the owner proxies they create. Moreover, they can combine the behaviours of any facet their WoN nodes support to build complex communication structures. Programmers should be enabled to create new facet contracts and implementations that can be plugged into any WoN node, thereby making the new behaviour available to all users within a short time after its inception. Thus, new ways of interaction between users can emerge without the need for a centralized platform.

We used a traditional commercial business case for illustrating the proposed framework. The design of the framework was influenced by empirical analysis of sharing communities which is beyond the scope of this paper. This fact, however, explains the current emphasis of natural language messaging and an understanding of transactions that is close to that of informal conversations. Further work is required for enabling the framework for traditional business transactions. We intend to evaluate the framework in the domain of sharing communities before adapting it to a commercial domain.

References

Aranda C. B., Corby O., Das S., Feigenbaum L., Gearon P., Glimm B., Harris S., Hawke S., Herman I., Humfrey N., Michaelis N., Ogbuji C., Perry M., Passant A., Polleres A., Prud'hommeaux E., Seaborne A., Williams G. T. (2013) SPARQL 1.1 Overview. http://www.w3.org/TR/sparql11-overview/. Last Access: 2016.06.06

Bizer C., Cyganiak R. (2014) RDF 1.1 TriG. http://www.w3.org/TR/trig/. Last Access: 2016.06.06

Brin S., Page L. (1998) The anatomy of a large-scale hypertextual Web search engine. In: Computer networks and ISDN systems 30(1), pp. 107–117

Carothers G. (2014) RDF 1.1 N-Quads. http://www.w3.org/TR/n-quads/. Last Access: 2016.06.06

Dalton B. (2013) Pseudonymity in social machines. In: Companion Publication of the IW3C2 WWW 2013 Conference. Rio de Janeiro, Brazil, pp. 897–900

Deakin N. (2003) JSR-000914 JavaTM Message Service (JMS) API. https://jcp.org/aboutJava/communityprocess/final/jsr914/index.html. Last Access: 2016.06.06

Hepp M. (2008) GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In: Proceedings of the 16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008). LNCS Vol. 5268. Acitrezza, Italy, pp. 332–347

Hohpe G., Woolf B. (2004) Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley, Boston, MA

Kasten A., Scherp A., Schauß P. (2014) A Framework for Iterative Signing of Graph Data on the Web. In: The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings. LNCS Vol. 8465, pp. 146–160

Kleedorfer F., Busch C. M. (2013) Beyond Data: Building a Web of Needs. In: Proceedings of the WWW2013 Workshop on Linked Data on the Web (LDOW 2013). Rio de Janeiro, Brazil http://ceur-ws.org/Vol-996/

Kleedorfer F., Busch C. M., Pichler C., Huemer C. (2014) The Case for the Web of Needs. In: Business Informatics (CBI), 2014 IEEE 16th Conference on Vol. 1. IEEE. Geneva, Switzerland, pp. 94–101

Manola F., Miller E. (2004) RDF Primer. http://www.w3.org/TR/rdf-primer/. Last Access: 2016/06/06

McBryan O. A. (1994) GENVL and WWWW: Tools for taming the web. In: Proceedings of the First International World Wide Web Conference Vol. 341. CERN. Geneva, Switzerland

Papazoglou M. (2008) Web services: principles and technology Pearson (ed.). Addison-Wesley, Harlow, England

Robinson I., Newcomer E. (2009) OASIS Web Services Business Activity Version 1.2. http://docs.oasis-open.org/ws-tx/wsba/2006/06. Last Access: 2016.06.06

Sambra A., Story H., Berners-Lee T. (2013) WebID 1.0 - Web Identity and Discovery. https://dvcs.w3.org/hg/WebID/raw-file/tip/spec/identity-respec.html. Last Access: 2016.06.06

Searls D. (2012) The intention economy: when customers take charge. Harvard Business Review Press, Boston, MA

Sporny M., Longley D., Kellog G., Lanthaler M., Lindström N. (2014) JSON-LD 1.0. http://www.w3.org/TR/json-ld/. Last Access: 2016.06.06

Tramp S., Frischmuth P., Ermilov T., Shekarpour S., Auer S. (2014) An architecture of a distributed semantic social network. In: Semantic Web 5(1), pp. 77–95

Windley P. J. (2011) The Live Web – Building Event-Based Connections in the Cloud. Course Technology, Boston, MA

Zimmermann A. (2014) RDF 1.1: On Semantics of RDF Datasets. http://www.w3.org/TR/rdf11-datasets/. Last Access: 2016.06.06